Moving Linux Applications to IA-64:
A Head Start

Table of Contents

Introduction

The benefits of porting existing Linux-based applications from the IA-32 environment to the IA-64 environment are substantial. Not only can you retain and leverage a proven source of value, but you also can enhance the usefulness of your applications with the added performance, scalability, and reliability of the IA-64 platform.

This document, which is tailored for C/C++ application developers using an IA-32/IA-64 cross-development system on an IA-32 Linux platform, provides a brief introduction to the porting process. In this document you will find information on the IA-64 Software Developers Kit, on the basic porting steps, and on commonly encountered warnings and errors.

The IA-64 Software Developers Kit

The first step in porting your application is to obtain the IA-64 Software Developers Kit (SDK) for the Linux platform. Go to http://www.linuxia64.org for information on where and how to download the SDK, which comes in a compressed file format.

Once you have done this, here are the steps required for you to install the SDK.

First, assume the file name is the following:

```
/usr/ia64-cygnus-linux/ia64LinuxSDK.tar.gz
```

Second, issue the following command:

```
tar -zxvf /usr/ia64-cygnus-linux/ia64LinuxSDK.tar.gz
```

Third, create a script /home/MyUsername/setIA64env to set up your environment variables:

```
#!/bin/bash

#This script sets up the IA-64 cross-compile
   environment variables

PATH=/usr/ia64-cygnus-linux/bin:/usr/ia64-
   cygnus-linux/ia64-cygnus-linux/bin:$PATH

#GCC_EXEC_PREFIX=ia64-cygnus-linux

LIBRARY_PATH=/usr/ia64-cygnus-linux/lib:/usr/ia64-
   cygnus-linux/ia64-cygnus-linux/lib:$LIBRARY_PATH

C_INCLUDE_PATH=/usr/ia64-cygnus-
   linux/include:/usr/ia64-cygnus-linux/ia64-cygnus-
   linux/include:$C_INCLUDE_PATH

CPLUS_INCLUDE_PATH=/usr/ia64-cygnus-
   linux/include:/usr/ia64-cygnus-linux/ia64-cygnus-
   linux/include:$CPLUS_INCLUDE_PATH

OBJC_INCLUDE_PATH=/usr/ia64-cygnus-
   linux/include:/usr/ia64-cygnus-linux/ia64-cygnus-
   linux/include:$OBJC_INCLUDE_PATH

PS1=IA64:$PS1

export PATH GCC_EXEC_PREFIX LIBRARY_PATH \

C_INCLUDE_PATH CPLUS_INCLUDE_PATH OBJC_INCLUDE_PATH PS1
```

Fourth, execute the IA-64 cross-development environment script:

```
. /home/MyUsername/setIA64env
```

At this point, you have completed installation of the IA-64 SDK. You are now ready to use the SDK for porting your code.

Basic Steps for Porting Your Application

Preparing for the compile

First, as with any other upgrade, you must ensure that your Linux-based application functions properly in its existing IA-32 environment. Do this by compiling the application under Linux/IA-32, recording and resolving any warnings or errors, and backing up the existing makefile(s). Repeat this process until the application compiles without warnings or errors.

Second, modify the makefile(s) for your Linux/IA-64 build, as shown in the following example. Note that this example assumes you are using the SDK and working in a cross-compile environment. If you are not using the SDK or you are not working in a cross-compile environment, you will not need to make these modifications.

```
# IA-64 modified makefile for cross-compile

CROSSCOMPILE = ia64-cygnus-linux-

TOOLPATH = /usr/ia64-cygnus-linux/bin/

CC=$(TOOLPATH)$(CROSSCOMPILE)gcc

LD=$(TOOLPATH)$(CROSSCOMPILE)ld

CPP=$(CC) -E

AR=$(TOOLPATH)$(CROSSCOMPILE)ar

NM=$(TOOLPATH)$(CROSSCOMPILE)nm

STRIP=$(TOOLPATH)$(CROSSCOMPILE)strip

AS=$(TOOLPATH)$(CROSSCOMPILE)as

IA64LIBS=-lc -ldb -lnss_files -lnss_dns -lresolv

IA64CFLAGS= -O2

IA64INC= -I /usr/ia64-cygnus-linux/ia64-cygnus-
   linux/include -static

IA64LIBDIR = -L /usr/ia64-cygnus-linux/ia64-cygnus-
   linux/lib/

OBJDUMP=$(TOOLPATH)$(CROSSCOMPILE)objdump

GENKSYMS=/sbin/genksyms

INCLUDEROOT = /usr/ia64-cygnus-linux/ia64-cygnus-
   linux/include

INCLUDE = ${IA64INC}

CFLAGS = ${IA64CFLAGS} ${INCLUDE}

LDFLAGS = ${IA64LDFLAGS}

LIBS = ${IA64LIBS}
```

Performing the compile

Now you are ready to compile the application in the Linux/IA-64 environment.

First, remove all the object files to ensure that a recompile occurs, and save the IA-64 warnings in a text file:

```
make -f makefilename clean

make -f makefilename > warnings
```

The warnings text file will provide a base reference that is necessary because subsequent warnings will vary as you make changes to the source code.

Second, resolve the IA-64 warnings by making necessary changes to the code. Note that these changes should not affect the functionality of the application in an IA-32 environment. Repeat this step until you have resolved all warnings.

Third, run regression tests on the IA-32 ___ to ensure that the changes you made did not "break" anything. Modify your changes as necessary and repeat the regression tests until you are assured that the ___ runs cleanly.

That's it. Now you have a ___ that you can recompile and retest when the operating system and production compilers are available for actual IA-64 hardware.

Commonly Encountered Warnings and Errors

As you may be aware, one of the biggest challenges in porting from the IA-32 environment to the IA-64 environment involves data models. Most of the warnings and errors you encounter during porting will stem from the fact that a given C/C++ application in the Linux/IA-32 environment uses the ILP32 data model, and the same application in the Linux/IA-64 environment uses the LP64 data model.

These data models differ in that in ILP32 the data types int, long, and pointer are 32 bits in length, and in LP64 int is 32 bits and long and pointer are 64 bits. Because of this difference, you are likely to encounter a number of data-truncation problems during porting.

Here are three common examples:

1. Variables receiving longs

Before porting:

```
long l;                     //l is a 64-bit data type
int i;                      //i is a 32-bit data type
long func(long l);      //func returns a 64-bit value
i = (int)l + func(l);   //64-bit to 32-bit assignment
```

After porting:

```
long l;                     //l is a 64-bit data type
size_t i;                   //i is a 64-bit data type
long func(long l);      //func returns a 64-bit value
i = l + func(l);        //64-bit to 64-bit assignment
```

2. Variables receiving pointers

Before porting:

```
char p;                          //&p is a 64-bit value
int i = &p              //64-bit to 32-bit assignment
```

After porting:

```
char p;        //&p is a 64-bit value
uintptr_t i = &p        //64-bit to 64-bit assignment
```

3. Calling printf with conflicting conversion type specifiers

Before porting:

```
printf("%x",ptr_value);                    //%x is 32-bit and will truncate the pointer
```

After porting:

```
printf("%p",ptr_value);                 //change to %p to display the full pointer value
```

For more information

This document is meant to serve as an introduction for developers porting existing Linux-based applications from the IA-32 environment to the IA-64 environment. Ideally, it will help to get you started on your porting efforts so that when IA-64 hardware is available your applications will be ready to compile and run.

For more information on Linux/IA-64 and the collaborative initiative to port Linux to that architecture and release it to Open Source under the GPL, go to http://www.linuxia64.org. For more information on the IA-64, go to http://developer.intel.com/design/ia-64/index.htm.